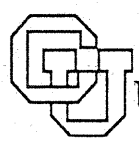
SUPRENUM:
PERSPECTIVES AND PERFORMANCE

Oliver A. McBryan

CU-CS-718-94



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

SUPRENUM: PERSPECTIVES AND PERFORMANCE

Oliver A. McBryan

CU-CS-718-94 1994

Department of Computer Science
University of Colorado at Boulder
Campus Box 430
Boulder, Colorado 80309-0430 USA

SUPRENUM: Perspectives and Performance*†

Oliver A. McBryan
Dept. of Computer Science
University of Colorado
Boulder, CO 80309.

Email: mcbryan@cs.colorado.edu

Abstract

We describe our impressions of the SUPRENUM project and of its primary supercomputer result, the Suprenum-1 prototype. We comment on the significance of the architecture, its role among contemporary systems and its relevance to current systems. We similarly discuss the SUPRENUM software and its impact on distributed systems. Finally we discuss the successes and failures observed throughout this exciting project and relate these to the organizational decisions on which SUPRENUM was based.

As an illustration of Suprenum-1 capabilities, we describe the implementation of a fluid dynamical benchmark on the 256 node Suprenum-1 parallel computer. The benchmark, the Shallow Water Equations, is frequently used as a model for both oceanographic and atmospheric circulation. We describe the steps involved in implementing the algorithm on the Suprenum-1 and we provide details of performance obtained. For such regular grid-based algorithms the system delivers a very impressive fraction (25%) of its theoretical peak rate of 5 Gflops.

Keywords: SUPRENUM, MPP, MIMD, parallel, supercomputer, performance, atmospheric, shallow water, architecture, software, message passing.

---

*Research supported in part by NSF Grand Challenges Applications Group grant ASC-9217394 and by NASA HPCC Group Grant NAG5-2218.
† To appear in Parallel Computing, October 1994.

TABLE OF CONTENTS

SUPRENUM: Perspectives and Performance*†

Oliver A. McBryan
Dept. of Computer Science
University of Colorado
Boulder, CO 80309.

Email: mcbryan@cs.colorado.edu

1. Introduction

The SUPRENUM project began in 1985 with the initial design formulation and continued until 1990 when a fully configured 256-node prototype Suprenum-1 machine was available. We have frequently visited both the GMD and SUPRENUM GmbH during this period and observed the project closely as it developed. During the same period we have also been a user of almost all other widely available MPP systems, and have been close to or involved in development of several of these.

We review the SUPRENUM project from this perspective. In Section 2 we discuss the origins, organization, successes and failures of SUPRENUM. We also provide an analysis of the project plan and indicate how it impacted the eventual outcome of the project. Section 3 describes the Suprenum-1 hardware and software. This section is oriented towards highlighting those areas where SUPRENUM broke new ground, or where it failed to achieve expectations.

To provide a more objective view we also present, in sections 4 and 5, measurements of Suprenum-1 performance and comparisons to its contemporary systems. We will describe simulations of the shallow Water Equations which attained 1.28 Gflops, and 25% of peak CPU rate, on a 256 processor Suprenum-1 computer. We also provide comparisons to the same application implemented on many of the other contemporary MPP systems.

In our discussion of Suprenum-1 hardware and software we will write in the present tense, although by publication time Suprenum machines may well no longer be running. We will use the upper case SUPRENUM name to denote the project as a whole and the lower case Suprenum name to refer to the Suprenum-1 prototype hardware and software.

---

*Research supported in part by NSF Grand Challenges Applications Group grant ASC-9217394 and by NASA HPCC Group Grant NAG5-2218.
† To appear in Parallel Computing, October 1994.

2. The SUPRENUM Project

2.1. SUPRENUM Origins and Organization

The SUPRENUM project had its origins in several groups who together provided complementary sources of expertise for the project. The initial group in this regard was the Numerical Computation Group from GMD, St.-Augustin, led by Dr. U. Trottenberg. This group had long experience in scientific supercomputing and initial experience with parallel computing. The group became interested in designing a parallel architecture that would be good for the multigrid algorithms that were then becoming well accepted for PDE solution. In 1984 the German government proposed a union of this group with a computer systems group in Erlangen. Later the Architecture and Systems group from GMD First in Berlin, led by Dr. W. Giloi, became interested in the project. These groups together created the original SUPRENUM vision. After an initial study phase in 1985, the project was formally initiated with a large grant from the German government. The government funding was strongly motivated by the desire to make sure that Germany remained at the forefront in advanced computing technologies.

The SUPRENUM mandate accompanying the funding was to create a project that included both a research and a commercial side. For this reason a company, SUPRENUM GmbH, was founded in Bonn. The SUPRENUM GmbH charge was to manage the whole enterprise, to contribute to the software effort, to coordinate software developments, and to exploit and market the results of the project. The commercial goal required that companies with manufacturing expertise be involved. The research aspects required that various university and government research laboratories should participate. The final team consisted of about 15 groups from institutions all over Germany, including several large companies as well as the small SUPRENUM GmbH.

2.2. SUPRENUM: Success or Failure?

There is no doubt that the research side of SUPRENUM was very successful. Some of the outputs are by now household words in the parallel computing area - for example PARMACS. Furthermore the project certainly achieved the major goal of moving Germany into the forefront of supercomputing developments. Many scientists trained in SUPRENUM have gone on to run other successful projects, or to develop machines and software in industry. Germany now has a large number of applications scientists who understand the issues involved in parallelizing real applications.

On the other hand, SUPRENUM hardware was not a commercial success with the result that the original SUPRENUM GmbH is no longer actively in business as a supercomputer vendor. Effectively, SUPRENUM GmbH, through a management buyout, has been transformed into a software company, Pallas GmbH,

which specializes in software for parallel systems. The Suprenum-1 supercomputer was produced and by 1990, five systems had been delivered. However the cost per Mflops was too high in relationship to competitors. Additionally there was no incentive to buy one because no successor machine, or line of machines, was ever announced. While concrete plans were developed for Suprenum-1+ and Suprenum-2 machines, no funding was ever available to develop either. Furthermore the prototypes were unstable, especially with respect to system software (e.g. the compiler) and certain hardware components (node boards failed randomly) and were not generally usable without some further effort. Unlike the situation in the USA, there was no German program to acquire significant numbers of Suprenum machines to be located at research centers.

SUPRENUM has attracted considerable negative press, sometimes even with suggestions that large sums of money were wasted because it failed. This is an inappropriate conclusion because it presupposes that SUPRENUM failed. Certainly the hope that the SUPRENUM experiment would create a successful German supercomputer company was not fulfilled. However the project as a whole was a success. All of the main software and hardware design goals were achieved. A working prototype was developed that for some months was the fastest MIMD MPP in the world. Trained scientists and engineers have spread from SUPRENUM to many other organizations in Germany, and a whole generation of students has been introduced to parallel processing.

Finally the SUPRENUM project has spun off many successful enterprises. As examples we cite: GENESIS, Pallas GmbH, Manna, PPPE and RAPS. Pallas in fact can be seen as a continuation of all of the software aspects of SUPRENUM, and as such shows that this part of SUPRENUM was commercially successful. The GMD FIRST project Manna is similarly a continuation of the operating system and some of the architecture aspects of SUPRENUM, again very successful, although this time in a research environment. Also the Meiko CS-2 machine, originally developed within GENESIS, involves many elements of the Suprenum-2 design from SUPRENUM, and indeed there were serious plans at one point to merge Meiko and SUPRENUM. Unfortunately this concept was ultimately rejected by the shareholders of SUPRENUM GmbH, who at that time also decided to withdraw from SUPRENUM. Finally the applications side of SUPRENUM evolved into GENESIS, and now PPPE and RAPS, so that again this aspect of SUPRENUM has shown itself to be of long-term viability.

Taking into account all of these achievements across a broad spectrum of computing technology, one can only conclude that SUPRENUM was highly successful, even while not achieving all of the goals originally established by the government.

2.3. SUPRENUM: An Outsiders Analysis

The "failure" of SUPRENUM was already determined quite early in the project by the requirement to have both a large distributed research aspect and a

commercial aspect. This immediately placed SUPRENUM in competition with many other MPP vendors such as Thinking Machines Corp., and Intel and even with vector supercomputer manufacturers such as CRAY Research. No successful commercial computer project has been developed by a team of 15 disparate groups, scattered throughout a country, with vastly different timelines and concepts of deliverability. The competitors typically develop a machine within a single building, with software, hardware, marketing and sales teams all in close and continuous contact.

In fact this was not the original view of SUPRENUM as proposed initially by Dr. U. Trottenberg and Dr. N. Szyperski, which was based on two phases. The first phase (3 years) was to be a research project, with some substantial industrial involvement, followed by an industry evaluation. The result of the evaluation was to be to either cancel the project, or to begin a second phase (2 years) as an industrial project, with some substantial research involvement. This proposal was accepted by the German ministry as an interesting experiment. However, due to a change in responsibilities in the ministry, the switch from Phase 1 to Phase 2 was never realized. Therefore after the first phase the project became a strange mixture of research and commercially oriented activities.

The SUPRENUM project model is typical of large consortium research projects - and this is indeed exactly where it was successful. Commercial success was further stymied by the absence of follow on funding for a second generation system. In fact even the Suprenum-1 prototype ran into problems just as it was about to become usable - funding dried up, when only about a man-year of extra effort could have enormously improved the system by fixing known serious compiler bugs. It is not so well-known that the German government provided no funds to Suprenum partners after 1989, long before the development effort was complete.

The two industrial shareholder companies that were partners in SUPRENUM GmbH were not really a great asset. To some extent they consumed more resources than they gave, and were in many ways a factor in failure of the effort, particularly because they were not willing to continue to support the effort at the critical point when development of a production quality Suprenum-2 machine was considered. In this respect, later EU Esprit projects were better planned because these typically require 50% matching from corporate partners, so that participating companies tend to be highly committed since they are spending their own funds. It seems unfortunate that a very strong company such as Siemens could not have taken over the primary role of developing, or at least funding development of a commercial Suprenum-2 product, after the initial government funding had created the Suprenum-1 prototype.

Saddled with such unrealistic conditions for commercial success, it is astounding that SUPRENUM was able to accomplish as much as it did. The final product was for a period in 1992 the fastest MIMD supercomputer in the world.

2.4. Conclusions

The clearest point to be learned from SUPRENUM is the importance of having a well-defined, focused and consistent plan for any such project which includes real product development. A project cannot be all things to everyone - e.g. both a university research project and a competitive commercial project simultaneously. High-tech projects have further constraints involving rapidly changing technology and the associated need to honor strict timelines, and these require an even narrower focus in order to bring the project to a successful conclusion. Had SUPRENUM been organized as an initial research effort followed by a largely commercial product phase, the outcome might have been quite different.

3. The Suprenum-1 SUPERCOMPUTER

3.1. Suprenum-1 Hardware

The Suprenum-1 computer prototype couples up to 16 processor clusters with a network of 200 Mbit/sec busses. The busses were intended to be arranged as a rectangular grid with 4 horizontal and 4 vertical busses, although other configurations have also been employed (see below). Each cluster consists of 16 processors connected by a fast bus, along with I/O devices for communication to the global bus grid and to disk and host computers. There can be a dedicated disk for each cluster. Individual processors can deliver up to 20 Mflops (64-bit chained) or 10 Mflops (64-bit unchained) of computing power and support 8 Mbytes of memory. The high bandwidth of the bus network makes this an interesting machine for a wide range of applications, including those requiring long-range communication. No more than four communication steps are ever required between remote nodes, with four steps needed only if both a horizontal and a vertical bus must be traversed.

While Suprenum clusters are well defined by their interconnection bus, the connectivity between clusters is modifiable by rewiring the connections appropriately. In principle this is simple, although in practice it turns out to be a major undertaking because there are severe physical constraints on the length of the buses involved, plus the fact that each bus must actually connect to form a ring. Each ring must visit from 4 to 6 clusters. During 1991, the Suprenum-1 clusters were connected in a simple ring (actually four parallel rings, although it was not possible to fully utilize the parallelism). In January 1992 the Suprenum-1 was re-configured as a full double matrix of busses. In June 1992 the Suprenum-1 topology was changed to provide a topology where each cluster has a direct connection to every other one so that all communication operations required at most three steps. We have used the machine in all of these configurations. Our measurements

indicated that the latter interconnection network is best as this provided the best overall performance from the three interconnection schemes which were studied.

Suprenum-1 hardware is generally about as reliable as similar products from other vendors. The system can sustain prolonged computations at times before developing one or another error state. In the most serious of these, node boards randomly fail with a mysterious and not fully understood condition called *'Trap 71"*, which renders the boards permanently damaged. (This problem may have disappeared recently when the 256 node system was split into two 128 node systems)

3.2. Suprenum-1 Software

Suprenum-1 software was developed on many levels:

Operating System

Vectorizing Compilers

Message Passing

Applications

The operating system for Suprenum-1 is PEACE, a new operating system developed specifically for the project. PEACE was designed from the start to support efficient low-latency message passing as well as multitasking. While PEACE appeared to be a satisfactory operating system, message latency never was as low as desired. Typical latency overheads are of order 1 millisecond. While asynchronous communication was a design goal for SUPRENUM, we were never able to overlap communication with computation on Suprenum-1 due to a mailbox conflict within PEACE.

The Suprenum nodes posed a major problem for compiler writers because of their inherent complexity. Furthermore the vector nodes involve various restrictions, of which the most serious is the need for very long vector lengths to achieve high performance. Frequently "very long" means length 1024 words or more. For data that is arranged as a square 2D or 3D grid this typically exceeds the memory of a Suprenum node. This means that in decomposing a large square or cubic grid into subgrids, with each subgrid assigned to a Suprenum node, it is essential that the subgrids not be square, but rather be elongated rectangles, with the longer length in the contiguous direction to maximize vector speed.

The node compiler supports both Fortran 77 and Fortran 90 array extensions. Suprenum-1 is the first MPP machine to support Fortran 77 on a vector node. The compiler includes powerful automatic vectorization which works well in our experience - i.e. it detects vectorizable code and generates efficient vector instructions. Unfortunately the compiler is in a development state only. There are many remaining bugs, some quite serious, and there is no funding available to fix these. . This, and the previously mentioned *"Trap 71"* hardware problem, are probably the most unfortunate features of the project, since they effectively render the machine unusable. No one with a large scientific application is ready to port it

to the machine because the compiler is so undependable. Small applications on the other hand run really well, once one finds any compiler bugs and recodes judiciously to avoid them. The failure by the government to allocate minimal extra funding to complete the compiler effort was probably a major mistake, because it prevented Suprenum-1 from being used effectively even in the research community (as noted earlier all Government support to SUPRENUM GmbH terminated in 1989).

Suprenum-1 is a message passing system. Unlike all other MIMD message passing systems, Suprenum-1 embedded message passing into the programming language. SUPRENUM Fortran uses message statements similar to Fortran I/O statements for this purpose. As a result the compiler is free to optimize the use of buffers and possibly even overlap communication with computation (this is not in fact done). As a simple example, an I/O statement can specify that a set of arrays and scalars of mixed data types be sent to another node, and the compiler takes care of packing these items into a single message to minimize startup overhead. In another Suprenum-1 Fortran extension, task control is provided at the language level. Addition of tasks to the language is essential if communication is to be fully represented at the language level, since the destination for messages is always another task.

One disadvantage of this approach is that SUPRENUM Fortran programs are not portable since no other system supports all of these various extensions. To overcome this problem, SUPRENUM software also supports a message passing library, callable from C or Fortran, providing an alternate to the use of Fortran task and messaging extensions. The compiler also generates these same calls so that a program can mix both styles of programming, although the interface is not well documented. This message passing system is very similar to other vendors products such as Intel NX.

Perhaps the most important aspect of SUPRENUM software was the emphasis on portable programming (this appears to contradict our previous paragraphs!). The SUPRENUM evaluation and application teams spent much effort in comparing rival MPP systems among each other and with Suprenum, and thus became familiar with competitors message passing systems (e.g. Intel NX). This led to a realization of the importance of message passing standards that would allow a single source code to run on all of these systems. The PARMACS portability platform evolved naturally from this start. The initial PARMACS developed from a collaboration between GMD and Argonne National Laboratory researchers, who had previously developed the P4 macro package. PARMACS significantly extended the functionality of P4 and focused on providing an easy to learn environment that would be highly portable, yet efficient on all platforms, and which provides support for grid-oriented computation. The success of PARMACS is shown by its wide adoption for most European MPP projects in the early 90's time frame. Furthermore PARMACS has had a strong influence on the new MPI message passing standard, which now includes the virtual processor topology aspects of PARMACS, among other features.

In addition to Fortran, SUPRENUM software was characterized by the best support for MIMD scientific applications to be found among the various distributed memory MIMD vendors. The effort invested in development of libraries of high-level grid and communication primitives greatly eases the effort of moving applications to the computer, and also provides substantial high-level portability to other systems, since the communication library can be implemented in terms of low level primitives on any distributed system. These developments arose naturally as the GMD group that initiated SUPRENUM were already involved in Multigrid development. They carried this experience to SUPRENUM with a realization of the special importance of 2D and 3D topologies in scientific computing. The COMLIB software is a high-level library, written on top of PARMACS, which allows users to efficiently organize MPP systems using virtual processor grids, and to map physical grids onto these processor grids.

3.3. Suprenum-1 Delivery and Performance

The first 32-processor prototype system was exhibited at the Hanover fair in April 1989, with the first 64-node system delivered in Fall 1989. The full-scale 256 processor system at the GMD became available in December 1990, although essential upgrades to the hardware continued into 1991. The full system has a 5 Gflops peak rating and has remarkably high realizable efficiency in appropriate applications, namely those where communication is relatively infrequent and where long vector lengths predominate.

We have benchmarked a number of applications on the system, of which the Shallow Water Equations (SWE) is a good example. The SWE is a standard model for atmospheric and oceanographic processes. Implementations of the algorithm have been used as benchmarks for vector and parallel supercomputer performance for many years. thus allowing effective comparisons among machines. The algorithms involved are typical of all explicit regular grid-based CFD codes, or of implicit codes that use iterative solution methods.

The rest of this paper will focus on our measurements of SWE performance on Suprenum-1. To summarize, we achieved 1.28 Gflops realized speed on large grids, or about 25% of peak performance. This is very high compared to most other MPP systems and indicates that Suprenum-1 was fundamentally well designed at both the hardware and software level.

4. The Shallow Water Equations Benchmark

As an example of the capabilities of the Suprenum-1 system, we describe the implementation of a standard two-dimensional atmospheric model - the Shallow Water Equations - on the machine. These equations provide a primitive but useful model of the dynamics of the atmosphere. Because the model is simple, yet captures features typical of more complex codes, the model is frequently used in the

atmospheric sciences community to benchmark computers [1-7]. Many of the results described here are presented in more detail in [5-7]. The SWE model has also been extensively analyzed mathematically and numerically [8-9].

The Shallow Water Equations, without a Coriolis force term, take the form:

$$\partial u / \partial t - \varsigma v + \partial H / \partial x = 0,$$
$$\partial v / \partial t - \varsigma u + \partial H / \partial y = 0,$$
$$\partial P / \partial t + \partial P u / \partial x + \partial P v / \partial y = 0,$$

where u and v are the velocity components in the x and y directions, P is pressure, ς is the vorticity: $\varsigma = \partial v / \partial x - \partial u / \partial y$, and H, related to the height field, is given by: $H = P + (u^2 + v^2)/2$. It is required to solve these equations in a rectangle $a \le x \le b$. $c \le y \le d$. Periodic boundary conditions are imposed on u, v, P, each of which satisfies $f(x+b, y) = f(x+a, y)$, $f(x, y+d) = f(x, y+c)$.

A scaling of the equations results in a slightly simpler format. Introduce mass fluxes $U = Pu, V = Pv$ and the potential velocity $Z = \varsigma / P$, in terms of which the equations reduce to:

$$\frac{\partial U}{\partial t} - ZV + \frac{\partial H}{\partial x} = 0,$$
$$\frac{\partial V}{\partial t} + ZU + \frac{\partial H}{\partial y} = 0,$$
$$\frac{\partial P}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0.$$

4.1. Discretization

The SWE are usually discretized on a rectangular staggered grid with periodic boundary conditions. The variables P and H have integer subscripts, Z has half-integer subscripts, U has integer and half-integer subscripts, and V has half-integer and integer subscripts respectively.

Initial conditions are chosen to satisfy $\nabla \cdot v = 0$ at all times. We time difference the equations using the Leap-frog method. We then apply a time filter to avoid weak instabilities inherent in the Leap-frog scheme:

$$F^{(n)} = f^{(n)} + \alpha(f^{(n+1)} - 2f^{(n)} + f^{(n-1)}),$$

where α is a filtering parameter. The filtered values of the variables at the previous time-step are used in computing new values at the next time-step. For a complete description of this discretization we refer to [1].

4.2. Serial Fortran Implementation

The Fortran code implementing the above algorithm involves a 2D rectangular grid with variables: *u(i,j), v(i,j), p(i,j), z(i,j), psi(i,j), h(i,j)*. There are three main loops, two corresponding to the Leap-frog time propagation of various quantities, and one for the filtering step. Execution of these three loops completes a single time step, which is then repeated until the desired temporal simulation interval has been achieved. A typical code sequence, used in the updating of the *U*, *V* and *P* variables, is:

$$
\begin{aligned}
&do\ 10\ j = 1, My \\
&do\ 10\ i = 1, Mx \\
&\quad unew(i+1,j) = uold(i+1,j) + tdts8 * (z(i+1,j+1) + z(i+1,j)) * \\
&\qquad (cv(i+1,j+1) + cv(i,j+1) + cv(i,j) + cv(i+1,j)) - tdtsdx * (h(i+1,j) - h(i,j)) \\
&\quad vnew(i,j+1) = vold(i,j+1) - tdts8 * (z(i+1,j+1) + z(i,j+1)) * \\
&\qquad (cu(i+1,j+1) + cu(i,j+1) + cu(i,j) + cu(i+1,j)) - tdtsdy * (h(i,j+1) - h(i,j)) \\
&\quad pnew(i,j) = pold(i,j) - tdtsdx * (cu(i+1,j) - cu(i,j)) - tdtsdy * (cv(i,j+1) - cv(i,j)) \\
&10\ continue
\end{aligned}
$$

Here the various coefficients such as *tdts8* are constants, determined by the discretization. Each such loop is followed by code to implement the periodic boundary conditions. Excluding the boundary computations, the three major loops in a time step involve 65 arithmetic operations per grid point. Furthermore 14 physical variables must be stored per grid point, which significantly limits the largest grid size that can be accommodated in a single node.

4.3. Suprenum-1 Implementation

The SWE code was developed for a generic class of MIMD parallel computers, based on the assumption of a single process per node model. The code was developed and tested using a simulator for the generic model developed previously [10-12]. The simulator supports versions of the Intel iPSC NX communication protocols among others.

Suprenum supports a library interface allowing Intel NX communication interfaces to be utilized. It suffices to declare the main program of both the host and node processes to be Suprenum *tasks*, while the rest of each program may remain as a pure Intel iPSC program. This allowed the code to be ported and fully working within hours. The program ran immediately and gave correct results on the first try. This demonstrates the advantages of developing MIMD codes initially using simulators, and transferring to hardware only when the simulations are running correctly.

Since the code involves rectangular grid arrays, and a nine-point stencil, the parallelization of the code is straightforward. A logical mapping of the processors to

a two dimensional array is utilized. If $P = p_x p_y$, is a factorization of the number of processors P, then we regard the processors as arranged in a $p_x \times p_y$ logical grid. Large arrays representing physical variables (u, v, etc.) are then decomposed into equal sized rectangular blocks, with one block assigned to each processor. For simplicity we assume that the X and Y grid dimensions are exact multiples of the corresponding processor numbers p_x and p_y. Each such block is then stored in an array of the same shape, but which has an extra boundary row or column provided on each of the four sides. These extra boundary points are used to maintain copies of the true (i.e. interior) boundary points of the four neighboring processors. The three main loops of the time step are decomposed into equivalent loops performed by each processor on the interior points of the block assigned to that processor. Prior to each loop, the boundary values are updated by exchanging appropriate values between neighboring processors, following a synchronization to ensure that all neighbors have completed updates. All data for each side of a variable block may be exchanged in a single packet to minimize communication latency costs.

There is an essential simplification that occurs in the case that either p_x or p_y is 1 - in which case the logical rectangular processor array reduces to a line of processors. In this case two of the four communications required within each main loop are not needed, reducing substantially the communication overhead. Normally periodic boundary conditions require copying data between processors at opposite edges of the processor array. In the case that one or other of p_x or p_y is 1, the periodic boundary condition in the corresponding dimension may be implemented by in-memory copying, rather than by communication.

A final optimization of the communication structure was required to get the peak performance. Before each of the main loops in the algorithm, the boundary data for the various physical variables (P, U, V, Z, H) used in that loop need to be copied from neighboring processors. Typically two or three variables are needed from a specific direction, although the number needed may depend on the direction. Because of the high communication startup cost of Suprenum-1 (close to 2 msecs), it is essential to limit the number of individual communication requests. This was accomplished by packaging several communications of different physical variables in a single direction into one large communication package. For some steps this reduced startup overhead by a factor of three. In the final implementation we also replaced the Intel iPSC communication calls for this one exchange operation by explicit calls to SUPRENUM Fortran equivalents, thereby saving an extra copying of each data array to a communication buffer. SUPRENUM Fortran supports explicit communication operations using a standard Fortran I/O control list syntax.

There is potential in the Shallow Water Equations to overlap communication with computation, provided the underlying hardware supports asynchronous communication modes. In this case one would begin each major loop by an asynchronous exchange of boundary data. Following this one executes the main body of the loop, however iterating only over the "interior points" of the subgrid. It is then necessary to await completion of the exchange operation, after which the

loop iteration may be completed on the outermost rows and columns. In principle such an approach can yield 100% computational efficiency - i.e. communication effects become negligible. We implemented such an algorithm on Suprenum-1. However due to inherent design aspects of the PEACE operating system we were unable to effectively use asynchronous communication in the current version of PEACE. This appeared to be due to a mailbox conflict.

5. Performance Results on Suprenum-1

All measurements were performed on a 256-processor Suprenum-1 system at the GMD, in St. Augustin, Germany. The Shallow Water Code was exactly the standard sequential code, modified only to take account of communication. No attempt was made to introduce Fortran 90 vectorization constructs, or to otherwise adapt the code to known features of the SUPRENUM compiler, other than in one line as mentioned above. The code was compiled with both the vectorizer and optimizer switches.

Because Suprenum nodes are vector processors, there is a substantial advantage to arranging the subgrids in each node such that the grid columns are as long as possible. In practice, Fortran columns longer than about 1024 words are not an advantage. This is because the vector registers are limited to a total of 7K words, and Shallow Water requires 7 registers for efficient code generation. Thus there is a tradeoff between register use and vector length. A larger vector register cache would have improved performance further.

In order to maximize computational efficiency (by minimizing communication words sent per Mflops), it is desirable to solve as large a problem as will fit in each node. For SWE, this turns out to be a problem with 32K grid points in a node which consumes approximately 6 Mbytes of the Mbytes of node memory. All measurements presented here utilize subgrids of maximal size, although their rectangular shape may vary. We maximize both vector performance and computational efficiency on a node by using a 32×1024 subgrid in each processor. To indicate the importance of preserving a long vector length we note that performance on a single node goes from 2.69 Mflops on a 128×256 grid to 5.33 Mflops on a 32×1024 grid, essentially a factor 2 improvement (see Table 1 below).

As discussed earlier, the number of communications per node can be reduced by a factor of two by choosing a one-dimensional processor grid, which may be aligned with either the X or Y axis. If the processors are in a line in the X direction, then the communication packets will be of size 1024 words (Y dimension of the subgrids) per variable, while if aligned along the Y axis, only 32 words are communicated per physical variable.

More generally we can expect lower performance as the subgrids tend towards a square shape, such as 128×256, due to the shorter vector lengths. Also using fully two-dimensional processor grids such as a 16×16 grid will double the number of

communications per node, resulting in poorer performance. All of these phenomena are illustrated in the measured results.

The final effect which we have studied is the influence of cluster interconnection topology on performance. The Suprenum-1 has been interconnected in three different ways as described in section 2 - ring, full matrix and full interconnect, and we have measured Shallow Water Equations performance in all cases. There is a significant dependence of performance on the topology used. For example the worst-case efficiency measured with the double matrix topology was 39% while with the full interconnect topology, the worst case efficiency is 70%. On the other hand for the most efficient (linear) cases, performance with the full connection topology is slightly worse, dropping from 96% to 94% efficiency. Clearly the advantages of the full interconnect topology outweigh the disadvantages. We give only the measured data for the full interconnect topology.

We present the measured results in Tables 1-4. The tables indicate the number of processors P, their arrangement as a logical $Px \times Py$ rectangular processor array, the computational domain size $Mx \times My$, the resulting computational efficiency and the Mflops generated. The computational efficiency in all cases is defined as:

$$E = T_{best}(1) / (PT(P)) ,$$

where $T(P)$ is the solution time with P processors and $T_{best}(1)$ is the best possible single-node performance with a subgrid of the same size but optimal shape.

Table 1 presents the effect of varying the grid shape in a single node. This demonstrates clearly the importance of maximizing vector length. Indeed the almost square 256×128 grid provides only 77% of the performance of the elongated 32×1024 grid with the same number of grid points. At the other extreme, the 1024×32 grid delivers only 43% of the performance of the 32×1024 grid.

TABLE 1: SINGLE NODE PERFORMANCE AS FUNCTION OF SHAPE						
P	Px	Py	Mx	My	Efficiency	Mflops
1	1	1	1024	32	0.430	2.29
1	1	1	256	128	0.768	4.09
1	1	1	128	256	0.883	4.71
1	1	1	64	512	0.955	5.09
1	1	1	32	1024	1.000	5.33

Table 2 describes the performance of Shallow Water on grids of optimal shape for the system. Each node contains an optimal 32×1024 grid and the processors are arranged in a line parallel to the Y direction in order to minimize communication.

TABLE 2: PROCESSOR GRID ALIGNED WITH Y AXIS						
P	Px	Py	Mx	My	Efficiency	Mflops
1	1	1	32	1024	1.000	5.33
2	1	2	32	2048	0.949	10.11
4	1	4	32	4096	0.948	20.21
8	1	8	32	8192	0.948	40.41
16	1	16	32	16384	0.946	80.61
32	1	32	32	32768	0.945	161.10
64	1	64	32	65536	0.947	322.74
128	1	128	32	131072	0.945	644.87
256	1	256	32	262144	0.940	1280.37

Table 3 is similar except that the processors are arranged in a line parallel to the X axis, resulting in more square grids, and slightly increased communication cost.

Table 3: PROCESSOR GRID ALIGNED WITH X AXIS						
P	Px	Py	Mx	My	Efficiency	Mflops
1	1	1	32	1024	1.000	5.33
2	2	1	64	1024	0.940	10.01
4	4	1	128	1024	0.934	19.89
8	8	1	256	1024	0.932	39.68
16	16	1	512	1024	0.931	79.29
32	32	1	1024	1024	0.919	156.76
64	64	1	2048	1024	0.920	313.24
128	128	1	4096	1024	0.912	621.83
256	256	1	8192	1024	0.905	1234.73

In Table 4, we compare the effect of varying the shape of the processor grid for 256 node computations. Each node is maintained at the optimal 32×1024 grid. The almost square 4096×2048 grid on a 128×2 processor array is seen to deliver 1117 Mflops. The alternative of creating a near square global grid from 256 near square subgrids would have yielded about 25% less performance as indicated by Table 1.

TABLE 4: 256-NODE PERFORMANCE AS FUNCTION OF PROCESSOR GRID						
P	Px	Py	Mx	My	Efficiency	Mflops
256	256	1	8192	1024	0.905	1234.73
256	128	2	4096	2048	0.820	1117.40
256	64	4	2048	4096	0.709	967.34
256	32	8	1024	8192	0.815	1110.72
256	16	16	512	16384	0.738	1007.87
256	8	32	256	32768	0.866	1180.16
256	4	64	128	65536	0.883	1202.52
256	2	128	64	131072	0.885	1206.58
256	1	256	32	262144	0.940	1280.37

5.1. A Comparison of 5 MPP Architectures

We have compared the Suprenum-1 performance with that on the CRAY X-MP and Y-MP computers, on the Intel iPSC/860 hypercube and on the CM-200 and CM-5 computers. Results are presented in Table 5. We compare Suprenum-1 only with machines that came out in the same generation or time period. Thus we omit measurements for example on the Intel Paragon or Cray Research T3D.

The CRAY-Y-MP with 8 processors runs the Shallow Water Equations at 1,530 Mflops on a 512^2 grid. The iPSC/860 performance was 543 Mflops on 128 nodes using the largest grid size that would fit in memory. Finally Suprenum-1 performance of 1280 Mflops was measured on a 256-node machine, again using the largest grid possible. The rationale for using such large grids is that the benchmark is a guide to behavior of realistic 3D applications where such grid sizes would be quite realistic. From the performance viewpoint, it is essential to use a maximal grid size per processor in order to minimize the interprocessor communication overheads on most machines. The CRAY measurements were made by Dr. R. Sato of the National Center for Atmospheric Research. The iPSC/860, CM-200, CM-5 and Suprenum results are described in more detail in [5-7,12].

TABLE 5: COMPARISON OF ARCHITECTURES

Machine	Processors	Grid Size	Mflops
CM-5 (32-bit)	1024	256M	23971
CM-5 (64-bit)	1024	64M	22139
CM-200 (32-bit)	2048	32M	8086
CM-200 (64-bit)	2048	16M	5249
CRAY Y-MP	8	256K	1530
CRAY X-MP	4	256K	560
Suprenum-1	256	8M	1280
Intel iPSC/860	128	2M	543

To relate Suprenum-1 to other systems, it is fair to say that for most of 1992 this system was probably the most powerful available MIMD system. However the SIMD CM-200 was far more powerful on SIMD problems. With the arrival of vector nodes for the Thinking Machines CM-5 computer, Suprenum-1 was no longer in this position in late 1992. Since then, faster systems such as the CRAY T3D, and IBM SP-2 have of course appeared. However for the purpose of this paper it seems most realistic to compare Suprenum-1 with contemporary machines of that time.

The performance measurements also should be qualified by the cost per Mflops of the different systems, which we have not considered in detail. However it

does appear that Suprenum-1 loses much of its performance advantage relative to the iPSC/860 if pricing is considered. This is due to the fact that the Suprenum nodes involve essentially more complex hardware (e.g. vector nodes) than the iPSC/860. The Suprenum node design was formulated long before the much cheaper i860 processor appeared.

REFERENCES

1. G.-R. Hoffmann, P.N. Swarztrauber, and R.A. Sweet, "Aspects of using multiprocessors for meteorological modeling," in Multiprocessing in Meteorological Models, ed. D. Snelling, pp. 126-195, Springer-Verlag, Berlin, 1988.
2. O. McBryan, "New Architectures: Performance Highlights and New Algorithms," Parallel Computing, vol. 7, pp. 477-499, North-Holland, 1988.
3. O. McBryan and R. Pozo, "Performance Evaluation of the Myrias SPS-2 Computer," CS Dept Technical Report CU-CS-505-90, University of Colorado, Boulder, 1990.
4. O. McBryan and R. Pozo, "Performance Evaluation of the Evans and Sutherland ES-1 Computer," CS Dept Technical Report CU-CS-506-90, University of Colorado, Boulder, 1990.
5. O. McBryan, "A Comparison of the Intel iPSC860 and Suprenum-1 Parallel Computers," Supercomputer, vol. 41, no. 1, pp. 6-17, 1991.
6. O. McBryan, "Scaling Performance of the Shallow Water Equations on the SUPRENUM-1," Supercomputer, vol. 53, pp. 4-14, Jan. 1993.
7. O. McBryan, "Performance of the Shallow Water Equations on the CM-200 and CM-5 Parallal Supercomputers," in Proceedings of the Fifth ECMWF Workshop on the use of Parallel Processors in Meteorology: Parallel Supercomputing in Atmospheric Science, ed. T. Kauranne, World Scientific, London, 1993.
8. R. Sadourny, "The dynamics of finite difference models of the shallow water equations," JAS, vol. 32, pp. 680-689, 1975.
9. G.L. Browning and H.-O. Kreiss, "Reduced systems for the shallow water equations," JAS, vol. 44, 1987.
10. R. Pozo, "Performance Modeling of Parallel Architectures for Scientific Computing," PhD Thesis, Department of Computer Science, University of Colorado at Boulder, 1991.
11. O. McBryan and E. Van de Velde, Hypercube Algorithms and Implementations, SIAM J. Sci. Stat. Comput., 8, pp. 227-287, 1987.

12. O. McBryan, "Software Issues at the User Interface," in Frontiers of Supercomputing II: A National Reassessment, ed. W.L. Thompson, University of Colorado CS Dept. Tech Report CU-CS-527-91 and MIT Press, 1994, to appear.